CMPE 561
Natural Language Processing
Application Project 2

Language Identification System

Ezgi Tekdemir
Beyza Gul Gurbuz

submitted to
Tunga Gungor

December 19, 2017

Introduction

For this project, we implemented a language identification system in Python using Discriminating between Similar Languages (DSL) Shared Task 2015 corpus. he analysis is based on character unigrams, meaning that the identification of a language is based on letter frequencies. The hypothesis is that each language either has unique characters or use the characters with unique probabilities.

In our corpus, there are 13 languages, each with 2000 sentences, so our corpus has a balanced distribution with 26000 sentences in total. The languages in question are Bulgarian, Bosnian, Czech, Argentine Spanish, Peninsular Spanish, Croatian, Indonesian, Macedonian, Malay, Brazilian Portuguese, European Portuguese, Slovak and Serbian.

Methods

We mainly tried two models to solve this problem.

Generative Modeling (Naive Bayes)

In this approach, we first train our model by calculating the letter probabilities for each language using Laplace Smoothing.

$$P(c_i|l) = \frac{number\ of\ times\ c_i\ appears\ in\ l + \ 1}{total\ number\ of\ characters\ in\ l + number\ of\ unique\ characters\ in\ l} \tag{1}$$

Then, for a sentence in the test set, we sum up the letter probabilities for each letter in the sentence and then multiply it with probability of that language in the corpus. Since there are same amount of sentences for each language, the probability for the language l is the same for all languages, $2000/26000 = 1/13$.

$$P(l|s) = (\sum_{i=1}^{n} P(c_i|l)) \cdot \frac{1}{13} \tag{2}$$

For a sentence, we do this for each language and say that the language with the highest probability is the predicted language for that sentence.

Discriminative Modeling (SVM)

For this approach, we used Cornell's SVM multiclass library. We used distinct letters that appear in the corpus as features and the existence of a character in that sentence as the corresponding value of that feature, i.e. the value of a feature is either 0 or 1.

While training, the system learns the important letters for each language. It uses a letter pool consisting of letters of all languages in the training set. Then, using this model, the system predicts the most likely language for that sentence.

As bonus, we added three other features to our SVM: count of letters, bigram letter frequencies and count of capital letters per sentence. The accuracy significantly increased with the first two features; however, the capital letter count decreased the accuracy of our predictions, so we decided to exclude it from the analysis at the end. We called the SVM with existence and count of unigram letter frequencies and count of bigram letter frequencies as *Super SVM*.

Evaluation

We evaluated the approaches used above with three types of metrics.

Accuracy

$$Accuracy = \frac{number\ of\ correct\ predictions}{number\ of\ sentences} \tag{3}$$

We calculated the accuracy for each language and for the entire test set.

Averaged Measures (Precision, Recall, F-measure)

In order to calculate these measures, the following values must be calculated first:

TP True Positive: Number of sentences whose language is L and predicted as L

FN False Negative: Number of sentences whose language is L but predicted as another language

FP False Positive: Number of sentences whose language is another language but predicted as L

Precision This measure answers the question of "Of all sentences that are labeled with language l, how many actually belong to language l?". It can be calculated as $\frac{TP}{TP+FP}$.

Recall This measure answers the question of "Of all sentences that truly belong to language l, how many did we label as language l?". It can be calculated as $\frac{TP}{TP+FN}$.

F-measure This is the weighted average of precision and recall. This is a better measure than accuracy if the cost of false positives and false negatives are very different. It can be calculated as $\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$.

Microaveraged Measures Gives equal weight to sentence language pairs, so F-measure uses averaged precision and recall values.

Macroaveraged Measures Gives equal weight to languages, so takes the average of precision and recall values regardless of sentence count.

At the end, we compare our approaches using these performance measures and find out which approach and features work best to identify the languages of sentences.

Program Interface

In order to run the program, Python3.5 must be installed. Put identifier.py in the same directory with the corpus file and rename the corpus file as "corpus.txt".

For the SVM part, download the source code from Cornell's SVM-Multiclass website. Then, go to svm-multiclass directory from the terminal and type:

```
> make
```

Wait for the code to compile. The executables svm-multiclass-learn and svm-multiclass-classify will be produced, which will be used in our program.

The program can be run in three modes: naive Bayes, unigram SVM and super SVM. Note that the program will not run without the mode argument.

In order to run the program in naive bayes mode, go to the directory of the python file and type:

```
> python identifier.py naive\_bayes
```

In order to run the program in SVM modes, path to the svm-multiclass directory must also be given as an argument.

For unigram SVM mode, type:

```
> python identifier.py unigram\_svm path/to/svm-multiclass
```

For super SVM mode, type:

```
> python identifier.py super\_svm path/to/svm-multiclass
```

Program Execution

It is required to give a mode to the program since there are three approaches that we use (naive bayes, unigram SVM and SVM with more features) to train our program.

For the naive bayes mode, The SVM modes also require the path to the svm-multiclass directory to run since the svm-multiclass-learn and svm-multiclass-classify executables used for training and testing the sentences are run inside our program using Python's *os* library. The commands executed are as follows:

```
> path/to/svm-multiclass/svm_multiclass_learn -c 1.0
      svm_train.txt svm_model.txt
> path/to/svm-multiclass/svm_multiclass_classify
      svm_test.txt svm_model.txt svm_output.txt
```

The first command is for training the system using the svm_train file we created in our program. This command produces svm_model file (in the directory of the Python file), which will be used in the next command to classify the test input. We chose the argument c (trade-off between training error and margin) to be 1.0 since we found out that the accuracy increases when this argument is higher. We do not allow the end-user to change this value.

The second command is for testing the system using the svm_test file we created in our program. This command uses the svm_model file and produces a file called svm_output. In this file, there is one line per a line in the svm_test file in the same order. The first value of each file is the predicted language number, and the following 13 numbers are the discriminant values for each of the language classes. In our program, we compare the first numbers of each lines of svm_test and svm_output to calculate the performance measures.

For all three modes, the produced output is of the same format. Accuracy, micro-averaged measures (Precision, Recall, F-measure) and macro-averaged measures (Precision, Recall, F-measure) are outputted to the terminal. In SVM modes, the multiclass library also produces an output before our output.

An example execution with unigram SVM mode is shown below:

```
Overall Accuracy: 43.07692307692308

Accuracy per Language:
pt-BR: 9.836065573770492
cz: 99.5
bs: 0.0
es-AR: 17.117117117117118
pt-PT: 35.5
sk: 6.310679611650485
sr: 0.0
hr: 55.55555555555556
bg: 99.05660377358491
mk: 100.0
my: 0.0
id: 88.71794871794872
es-ES: 46.63677130044843

Micro averaged Precision: 0.4307692307692308
Micro averaged Recall: 0.4307692307692308
Micro averaged F-measure: 0.43076923076923074

Macro averaged Precision: 0.48180332869146836
Macro averaged Recall: 0.4294082628077506
Macro averaged F-measure: 0.382090673270263
```

Figure 1: Example execution of naive bayes mode

Input and Output

As the input, DSL Shared Task 2015 corpus is used. In this file, there are 26000 sentences with their language labels each seperated with a new line character. A sentence and its language are seperated with multiple space characters. The labels are translated as follows:

bg Bulgarian

bs Bosnian

cz Czech

es-AR Argentine Spanish

es-ES Peninsular Spanish

hr Croatian

id Indonesian

mk Macedonian

my Malay

pt-BR Brazilian Portuguese

pt-PT European Portuguese

sk Slovak

sr Serbian

```
Балканите са доста сеизмично активен регион и няма нищо чудно, че
след „Фукушима” българите са се замислили за допълнителни мерки за
сигурност", коментира пред московския електронен вестник „Газета”
руски банков анализатор.        bg
```

Figure 2: An example Bulgarian sentence

The algorithms are applied on this corpus, by randomly separating the sentences as 90% training and 10% testing. Naive Bayes algorithm directly gives an output on the terminal. The SVM algorithms, however, require svm_train and svm_test files to run. These two files have the same structure. On each line, there is the language-id, features and values corresponding to

each sentence. In our case, there are 13 language-ids. In unigram SVM mode, there are 264 possible values since the unique character size summed up for all languages is 264.

language-id f1:v1 f2:v2 f3:v3...

For example, if the language-id for English is 2, character type is a feature and the count of those characters are the corresponding values, and the English alphabet is mapped to values from 1 to 26, the word "bee" would be represented with the following line:

2 2:1 5:2

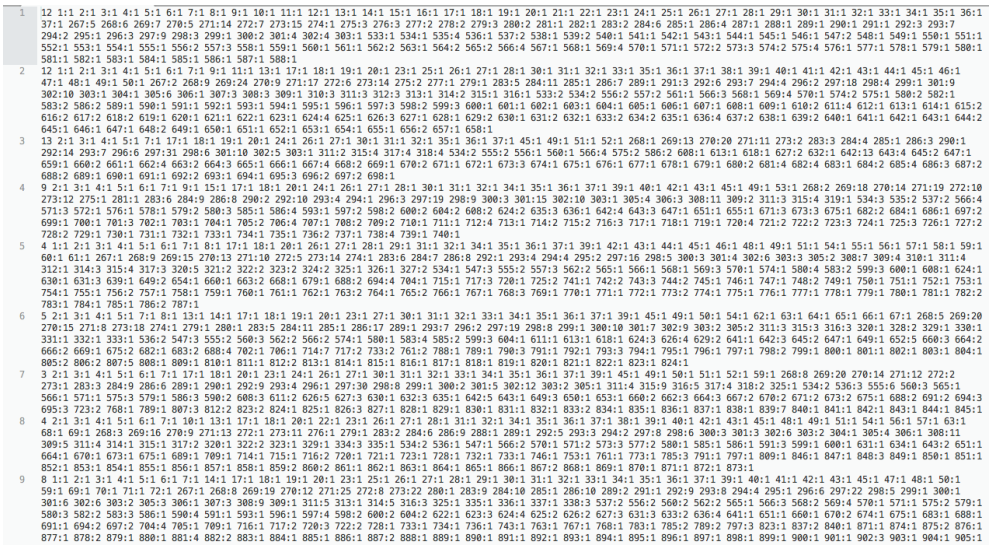

Figure 3: Example train file (same format with the test file)

In the svm_model file, the learned rule from the training using the regularization parameter c (in our case, 1.0) is stored.

Figure 4: Example model file

In addition to terminal output, SVM algorithms give an output file called svm_output.txt. In the svm_output file, there are 14 numbers per line. The first number denotes the language id and the other 13 are the discriminant values for each of the language classes (values of x $\cdot w_i$).

```
1 1.058748 0.513624 -0.343680 -0.289460 0.535259 -0.271806 -0.927521 0.439519 -0.125423 0.867802 0.046700 0.164319 -1.668080
1 0.970500 0.144677 0.387373 -0.052790 0.707876 -0.211918 0.509613 0.274895 0.255897 0.445447 0.022077 0.283738 -3.737385
1 3.986361 0.339907 -0.460615 -0.283309 1.317885 -0.312253 -0.291336 0.415400 -0.699486 0.557482 -0.194305 -0.213764 -4.161965
1 5.427884 0.615121 -0.536092 -0.732228 2.079143 -0.470376 -3.535466 1.248478 0.081740 1.431100 -0.336035 0.818377 -6.091645
1 2.446304 0.591722 0.273131 0.239190 0.957831 -0.411390 -0.827424 0.416147 -0.129849 0.180308 0.016111 0.173344 -3.925423
6 -2.046460 -1.372880 -0.908590 -1.190624 -1.305525 15.011465 -1.070947 -0.754543 -1.129117 -2.510448 0.175854 -0.866821 -2.031364
1 6.982110 0.051128 -0.641796 -0.284969 1.886108 -0.624168 -0.354003 0.441763 -0.826537 0.264245 -0.205284 -0.311152 -6.377444
13 0.539952 -0.632712 0.974619 -1.891523 -1.459629 -0.307100 -2.634197 0.678283 -0.461676 -1.744802 -0.219517 -0.282573 7.440877
1 3.211593 -0.174827 -0.388985 -0.224002 0.703785 -0.193468 -1.709679 0.633544 -0.045981 0.154416 -0.194152 0.089348 -1.861591
1 1.187408 0.875973 -0.132455 0.354725 0.603316 -0.202952 -0.270704 0.372595 0.018535 0.788137 -0.044437 0.131416 -3.681556
7 0.536807 -0.447892 -0.544095 -0.305357 -0.411441 -0.199837 5.248810 -0.358790 -0.604080 -0.781537 -0.054165 -0.508435 -1.569988
13 -0.339053 0.037010 0.650171 -0.265436 -0.619453 -0.256342 -1.287139 0.203086 -0.315439 -0.432882 0.036374 -0.180827 2.769929
1 2.972568 0.833958 0.742257 -0.478758 1.551686 -0.422608 -0.070436 0.797806 0.070735 0.675821 -0.087673 0.369711 -6.955065
8 -0.126860 0.338263 -0.287826 0.239593 -0.056388 -0.152388 -1.493063 1.164367 0.715411 -0.069309 -0.081206 0.872598 -1.063192
1 0.997820 0.040580 -0.001416 -0.029371 0.470262 -0.079866 0.446227 0.182377 -0.183965 0.250812 -0.045743 -0.094703 -1.953012
10 1.140321 1.149601 -0.342041 -0.082384 0.700001 -0.192935 0.529145 0.546878 -0.039823 1.361333 -0.107032 0.279500 -4.942563
3 -0.899070 0.340311 0.961493 0.893189 -0.240984 -0.080010 -0.612669 0.849670 0.752039 0.182624 -0.089430 0.612697 -2.669859
1 1.922431 0.544695 -0.330585 0.515755 0.454413 -0.335006 -1.258339 0.404009 -0.142353 0.031832 -0.065575 0.080216 -1.821492
4 -0.709777 -0.019122 -1.323385 3.266811 -0.782695 -0.187421 2.351431 0.123602 0.215948 -0.287083 -0.059961 0.194607 -2.782957
13 -0.490941 0.144880 0.875674 -0.966258 -0.580012 -0.131711 -1.382337 0.346568 -0.194365 -0.378938 -0.141477 0.022449 2.876469
9 -0.737940 0.244245 0.238789 0.206797 -0.191082 -0.121112 -0.706594 0.834344 1.106539 -0.151736 -0.091001 0.812683 -1.443934
2 -0.211260 1.311312 -0.006612 0.364959 0.235416 -0.140435 -2.759553 0.356290 0.297022 0.510357 -0.021672 0.413010 -0.348834
7 -0.073769 0.047666 -0.270783 -0.394111 -0.219322 -0.140280 3.148876 0.064358 -0.204150 -0.000205 -0.038825 -0.162104 -1.757352
9 -0.507726 0.391565 -0.525688 0.394936 -0.403565 -0.152637 -1.200666 0.823988 1.040513 -0.297300 -0.065776 0.772457 -0.270100
8 -0.960210 0.586274 -0.694375 0.414117 -0.389304 -0.196331 -0.293192 1.716922 1.128033 0.086552 -0.189068 1.202110 -2.411527
4 -0.051298 1.278124 -0.231562 1.538724 0.336049 -0.162492 -1.918144 0.379856 0.350053 0.733039 -0.027524 0.413221 -2.638046
6 -1.654180 -1.127785 -0.718330 -0.986036 -1.078393 7.343685 -0.865886 -0.654597 -0.905284 -2.043804 5.033987 -0.682030 -1.661348
1 1.157840 0.308436 0.149041 -0.211876 0.858694 -0.342383 -0.128177 -0.000050 -0.056727 0.675681 0.080539 0.172783 -2.663800
6 -2.048747 -1.474155 -0.959299 -1.288125 -1.435200 17.026497 -1.182861 -0.828464 -1.166161 -2.625998 -1.057946 -0.865834 -2.093707
1 2.512766 0.527390 -0.741557 0.002135 0.797532 -0.235353 -1.268173 0.320331 -0.641914 0.624015 -0.131727 -0.321771 -1.443676
6 -1.593631 -1.076447 -0.736691 -0.934637 -1.005330 9.698203 -0.848751 -0.577070 -0.880356 -1.960239 2.102019 -0.660373 -1.526696
8 -0.597048 1.051012 0.081846 0.794518 -0.207024 -0.275291 -2.442294 1.222201 0.952883 0.172871 0.034541 0.942640 -1.730854
```

Figure 5: Example output file

We use svm_output.txt and svm_test.txt as inputs to our program again to compare the predicted and correct languages and evaluate our results. We only use the language-ids to compare these files.

Finally, as the terminal output, we give overall accuracy, accuracy per language, micro averaged measures and macro averaged measures. The accuracies are given in percentage and precision, recall and F-measure values are between 0 and 1.

Program Structure

The structure of the language identification system consists of two parts which are identification models and functions that measures accuracy and metrics of these models. At the beginning of the program, the corpus data is randomly divided into two parts as training and test data such that training data includes 90% of the data while test data includes %10. The training and test sets are kept in list of tuples such that each tuple includes the language and the sentence written in that language.

Approaches

We used three approaches in our language identification system. The approach to be used can be chosen by giving it as an argument to the program. Note that the space character is omitted in our analysis since we think that it does not give any valuable information about a language.

Naive Bayes

In the naive bayes model, all sentences of each language are concatenated and acted as one big string. For each character in a language, the probability of being in that language is calculated with laplace smoothing method for the training part. The result is kept in a dictionary whose key is language, and value is a dictionary which includes characters as key and their probabilities as value. In the test part, the probabilities of characters found in the training part in each sentence of test set are added for each language. The result is divided to the number of languages. The language with the highest probability is chosen for that test sentence. The results for each sentences are put in a dictionary whose key is a tuple mentioned above and value is a descending ordered sorted dictionary which contains the probability of being the language of that sentence for each language.

SVM

We used $SVM^{multiclass}$ library for SVM learning model. The inputs given that library are prepared in SVM part. Unlike naive bayes, we did not concatenate sentences for each language and determined features and their values for all sentences. Since the library only accepts positive integers as class and feature id, we give unique ids to all languages(classes) and features and put them in dictionaries. After creating input files, for both training and test sets, shell scripts that execute svm_multiclass_learn and svm_multiclass_classify

are run. The original languages of test sentences and the first line of the output file which includes guessed languages of test sentences are put in a list of tuples.

Unigram SVM In the unigram SVM, we selected the existence of characters in each sentence as a feature. We gave a feature id to each character in the corpus, and for each character, if it is in a sentence, that character is given as a feature with value 1 for that sentence.

Super SVM In the super SVM, in addition to unigram SVM, we added the number of characters and the number of bigrams in each sentence as features. The bigrams in the corpus are kept in a dictionary with keys as bigrams, and values as ids. The values of character and bigram features are selected as their number of existence for each sentence.

Performance Measures

We created two functions to measure the performances of the learning models.

Accuracy

The accuracy function takes tuple list of original and guessed languages as input. It calculates the proportion of all test sentences that are guessed correctly. The accuracy of each language is also calculated and they are returned.

Metrics

The metrics function takes tuple list of original and guessed languages as input. precision, recall, and F-measure values which are mentioned in evaluation section for both micro and macro averaged cases are calculated. They are returned as values of a dictionary whose keys indicate if they are micro or macro.

Results

Table 1: Accuracies

	Naive_Bayes	**unigram SVM**	**super SVM**
Overall Accuracy	**32.65384**	**44.73076**	**59.30769**
Accuracy of bg	0.96618	100.0	67.59776
Accuracy of bs	23.67149	89.78494	26.0
Accuracy of cz	0.0	100.0	98.93617
Accuracy of es-AR	0.0	0.0	1.53846
Accuracy of es-ES	95.83333	0.0	96.78899
Accuracy of hr	27.02702	0.0	55.72139
Accuracy of id	0.94339	95.43378	12.09302
Accuracy of mk	100.0	100.0	100.0
Accuracy of my	100.0	0.0	97.51243
Accuracy of pt-BR	0.0	0.0	91.87817
Accuracy of pt-PT	70.65217	27.52809	16.5
Accuracy of sk	0.0	38.80597	84.81675
Accuracy of sr	1.63043	0.0	23.52941

It is interesting to see that the accuracy for es-ES is around 95%, but es-AR is below 5%. We can say that the system classified the Spanish originated languages to es-ES. We can also clearly say that the languages which have distinct alphabets such as Bulgarian have very high accuracies, which also makes sense since those letters do not appear in any other language. It is also interesting to see that Czech has 0 accuracy in Naive Bayes whereas in SVM, the accuracies are nearly 100%. However, it should be also noted that the sentences are randomly seperated into training and test set, so if the test set does not include that language, accuracy would be zero.

It can be also seen that language-specifically, the unigram SVM has accuracies of zero. However, in super SVM, there is never a 0 accuracy, although there are accuracies near 1%.

In addition to these algorithms, we also tried to include number of capital letters as a feature too. The accuracy was better than Naive Bayes and unigram SVM approach; however, it was worse than the super SVM (it was around 50%). Hence, we decided to exclude that feature from our analysis.

Table 2: Metrics

	Naive_Bayes	**unigram SVM**	**super SVM**
Micro averaged Precision	0.32653	0.44730	0.59307
Micro averaged Recall	0.32653	0.44730	0.59307
Micro averaged F-measure	0.32653	0.44730	0.59307
Macro averaged Precision	0.29718	0.35139	0.60590
Macro averaged Recall	0.32363	0.42390	0.59454
Macro averaged F-measure	0.20307	0.34042	0.53361

Improvements and Extensions

In the project, we improved the accuracy of our identification system from approximately 40% to 60% by adding new features to svm input. Although this is a remarkable improvement, there could be more features which will increase the accuracy of the learning model.

We thought about adding word unigrams and average word length as features, but since we eliminated the space characters at the beginning of the program, it would be difficult. Therefore, we decided not to add them.

Difficulties Encountered

The first difficulty we encountered was because of the project description. In the description, we were told that the corpus is in UTF-8 format. However, we could not process the input properly before encoding it in UTF-16 format. After we discovered that the corpus is in actually UTF-16 format, we could read the file properly.

Implementing our code, we did not face with many issues but we had to choose what we wanted to do in corner cases. For example, when we were implementing averaged measures (precision, recall and F), we got float division by zero error when TP_i and FP_i values or TP_i and FN_i values are zero at the same time in Naive Bayes. We decided to set the corresponding precision, recall and F values to zero if their denominator is zero.

We also had a "key not found" error when we tried to get the letter probability of a character in a test sentence when that character does not exist in the training set. In this case, instead of adding zero to the probability sum of $P(c_i|l)$, we added $\frac{1}{total\ chars\ in\ l + unique\ chars\ in\ l}$ since we are applying laplace smoothing.

Conclusion

In our project, we used two models, one discriminative and one generative, to implement a language identification system. We compared these models with some performance measures to see that SVM works better than Naive Bayes. We could improve the accuracies significantly in SVM when we added the count of unigram letters and bigram letters as seperate features.

Bibliography

[1] **[Cornell's SVM-multiclass library]**
https://www.cs.cornell.edu/people/tj/svm$_l$*ight*/*svm*$_m$*ulticlass.html*

[2] **[Accuracy, precision, recall, F-measure]**
http://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/